COMSC-210 Lab 5
Big Data

GOOD PROGRAMMING PRACTICES *show / hide*

ABOUT THIS ASSIGNMENT
This assignment is the first in a series that applies data structure concepts to real-life problems involving big data. We solve memory issues by using the expandable DynamicArray template instead of our StaticArray, but time issues are not addressed... yet.

After you complete this lab assignment, post the required files to the COMSC server so that you receive the proper credit for this 50-point lab. Your posted files are due at midnight of the evening of the due date indicated in the course outline. Use the "Submit your FA2015 work" link on the class website to post your file for this lab, using the **lab6** folder provided.

LAB 5a: Applying A Data Structure To A Big Data Application `[ DvcSchedule5a.cpp ]`
Purpose. In this lab you will have your first experience with manipulating big data. The data is extracted from the DVC database of all class sections offered at DVC since the Fall 2001 semester. Your program is to list all of the subject codes (like COMSC, MATH, PHYS, etc), and include for each subject code the count of classes (e.g., MATH, 4514 classes).

Requirements. Write **`DvcSchedule5a.cpp`** to read and parse the 70,000 line dvc-schedule.txt text file, and find each subject code in the file. (Refer to the lecture topic 5 notes for how to parse this file, and refer to Burns' Comsc-110 textbook, chapter 10, for file-opening/closing syntax.) Output each code to the console screen, in alphabetical order, with the number of classes offered under that code. Use your own `DynamicArray` template from lab 4a. Do NOT use any STL containers.

Note -- the dvc-schedule.txt file may contain duplicate entries. The combination of a term and section number is supposed to be unique. A duplicate entry is when the same term and section number pair appear in more than one record. Do NOT count duplicates -- skip them. That means to count a duplicate entry only once, ignoring all others. You'll need some way to track what's been counted so that you don't count the same section for the same semester more than once. When you are done processing the input file, output HOW MANY DUPLICATES you found and skipped in the input file. Check that number with your classmates, because you should all come up with the same number.

At the end of your program, as the LAST thing it outputs, show the program's runtime in decimal seconds. Rounding is not necessary, but if you do choose to round, include at least two decimal digits. Include only the time to process the file and sort in order, but *not* the time to output the results. That is, (1) start timer, (2) open input file, (3) process input file in an EOF loop, (4) close input file, (5) sort, (6) stop timer, and (7) output the results. Use this to start the timer: `clock_t startTime = clock();` Use this to stop the timer: `double elapsedSeconds = (double)(clock() - startTime) / CLOCKS_PER_SEC;`, and then output `elapsedSeconds`, properly labeled. You'll need the appropriate C library for the definitions -- figure that out for yourself.

You can expect the runtime to be several minutes. So that you don't stare at a blinking cursor while you wait for results, add a "progress bar". To do so, count the number of lines read from the file. For every 1000 lines read, output a dot -- like this: `cout << '.';`. No `endl`, but you'll need `cout.flush();` to

force output out of the output buffer. After the EOF loop ends, output an `endl`, so that your output starts on a line *after* the line of dots. Or use some other method of indicating progress, as you prefer.

Follow the algorithm in the lecture notes to solve this.

Be careful! Don't just accept whatever counts that your program gives you. Make sure that your program gives the right answers for the input file used. Try using a much shortened version of the TXT file, for which you know exactly what to expect. Also try loading the TXT file into Excel -- sort the data in column A, and count for yourself to verify the results of your app.

Submit the CPP file to the class website for credit. Since you use your own `DynamicArray` template, there is no need to submit anything other than what you submitted already for lab 5a. Do **NOT** submit the very large input TXT file.

Program I/O. Input: Text file I/O: dvc-schedule.txt, located in the working folder (that is, no folder name in the open statement). Output: From the main program CPP only, console (`cout`) output only.

Example. (based on a previous year's version of the TXT file)

```
ADJUS, 557 sections
ADS, 206 sections
AET, 62 sections
ANTHR, 596 sections
ARABC, 13 sections
...
SPTUT, 12 sections
TAGLG, 8 sections
```

LAB 5b: Using A Data Structure As A Data Member `[ DvcSchedule5b.cpp ]`
Purpose. In this lab you will gain more experience with big data. The database is the same list of all class sections offered at DVC in the last 15 years, as you used in lab 5a above. Your program is to list all of the subject codes (like COMSC, MATH, PHYS, etc), and include for each subject code the count of *courses* (e.g., MATH, 47 course(s)). Then under each subject code, list each *course* (like Math 110) and the number of sections in the database (e.g., MATH-110, 588 section(s)).

Requirements. Write **`DvcSchedule5b.cpp`** to read and parse dvc-schedule.txt, and find each subject code in the file. Output each code to the console screen as a heading, in alphabetical order, with the number of *courses* offered under that code. Under each heading, list the course numbers belonging to that subject code, and the number of *sections* offered of that course. (The subject codes have to be in alphabetical order, but the courses do *not*).

Again, the dvc-schedule.txt file may contain duplicate entries. If you find an entry with the same *term* and same *section number* as another entry, it's a duplicate. Do NOT count duplicates.

The solution will be covered in the lecture notes!

Submit the CPP file to the class website for credit.

Program I/O. Input: Text file I/O: dvc-schedule.txt, located in the working folder (that is, no folder name in the open statement). Output: From the main program CPP only, console (`cout`) output AND output to an Excel XLS file.

Example. (based on a previous year's version of the TXT file)

```
ADJUS, 16 course(s)
  ADJUS-120, 191 section(s)
  ADJUS-121, 57 section(s)
  ADJUS-122, 40 section(s)
  ADJUS-130, 24 section(s)
  ADJUS-203, 20 section(s)
...
SPTUT, 1 course(s)
  SPTUT-020NC, 12 section(s)
TAGLG, 2 course(s)
  TAGLG-155, 5 section(s)
  TAGLG-156, 3 section(s)
```

In addition to console output, write the results to an Excel file named **DvcSchedule.xls**. Click HERE for syntax details. Organize the output as you wish, but make sure that the course titles (like COMSC-210) and their totals (like 45) are in separate columns.

How to pause a console program: *show / hide*

GRADING RUBRIC *show/hide*
